


Detecting concurrency errors in multi-threaded programs

Patent Number: ☐ EP0864975
Publication date: 1998-09-16
Inventor(s): SAVAGE STEFAN (US); BURROWS MICHAEL (US); NELSON CHARLES G (US); SOBALVARRO PATRICK G (US)
Applicant(s): DIGITAL EQUIPMENT CORP (US)
Requested Patent: ☐ JP10254716
Application Number: EP19980104160 19980309
Priority Number(s): US19970815979 19970310
IPC Classification: G06F11/00
EC Classification: G06F11/00A4
Equivalents: CA2231597, ☐ US6009269

Abstract

A computer implemented method detects concurrency errors in programs. Machine executable images of multiple program threads are instrumented to locate and replace instructions which effect concurrency states of the threads. Concurrency state information is recorded in a memory while the multiple threads are executing. The recorded concurrency state information is

analyzed, and inconsistent dynamic concurrency state transitions are reported as concurrency errors. 

Data supplied from the esp@cenet database - 12

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-254716

(43) 公開日 平成10年(1998) 9月25日

(51) IntCl.⁶

G 0 6 F 9/46
11/30
11/34

識別記号

3 4 0
3 0 5

F I

G 0 6 F 9/46
11/30
11/34

3 4 0 G
3 0 5 D
A

審査請求 未請求 請求項の数10 OL (全 12 頁)

(21) 出願番号 特願平10-58064

(22) 出願日 平成10年(1998) 3月10日

(31) 優先権主張番号 08/815979

(32) 優先日 1997年3月10日

(33) 優先権主張国 米国 (U S)

(71) 出願人 590002873

ディジタル イクイブメント コーポレイ
ション

アメリカ合衆国 マサチューセッツ州

01754-1418 メイナード パウダー ミ

ル ロード 111

(72) 発明者 マイケル バロース

アメリカ合衆国 カリフォルニア州

94303 バロ アルト ディヴィッド コ
ート 3143

(74) 代理人 弁理士 中村 稔 (外6名)

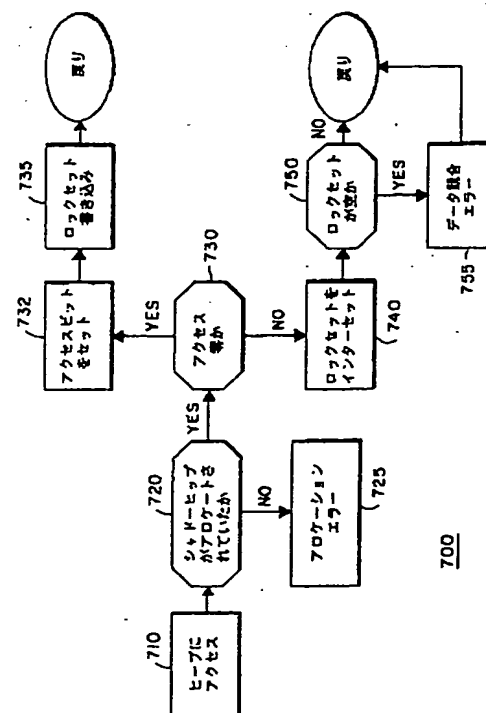
最終頁に続く

(54) 【発明の名称】 マルチスレッドッドプログラムにおけるコンカレントエラーの検出

(57) 【要約】

【課題】 プログラムスレッドをコンカレントに実行するに際し、コンカレントエラーを検出すること。

【解決手段】 コンピュータによって実施される方法は、プログラムにおけるコンカレントエラーを検出する。複数のプログラムスレッドの機械イメージは、スレッドのコンカレント状態をもたらす命令を捜し出し置き換えるようにインストールされる。複数のスレッドが実行されている間に、コンカレント状態情報はメモリに記録される。この記録されたコンカレント状態情報は、分析され、一致しない動的コンカレント状態遷移が、コンカレントエラーとして報告される。



【特許請求の範囲】

【請求項1】 プログラムにおけるコンカレントエラーを検出するためのコンピュータによって実施される方法であって、プログラムの複数のスレッドの機械語をインストールメンティングして、該スレッドのコンカレント状態をもたらす命令を捜し出して置き換え、前記複数のスレッドが実行されている間にメモリにコンカレント状態情報を記録し、該記録されたコンカレント状態情報を分析し、一致しない動的コンカレント状態遷移をコンカレントエラーとして報告することを含む請求項1記載の方法。

【請求項2】 ロックを獲得し解放する命令を捜し出し、該捜し出された命令を実行フローをインタセプトする命令と置き換え、実行されている間に前記複数のスレッドがロックを獲得し解放するオーダーを記録し、前記複数のロックがロックを獲得するオーダーにおいて一致しないコンカレント状態遷移をサイクルとして報告し、実行フローをインタセプトするときにモニタリングプロシージャをコールすることを含む請求項1記載の方法。

【請求項3】 各スレッドに対して、スレッドに関連した現在のロックセットを維持し、ロックが獲得されるときに前記ロックセットにそのロックを加え、前記ロックが解放されるときに前記ロックセットからそのロックを削除し、前記ロックが獲得されるオーダーをロックログに記憶させ、新しいロックが獲得されるときに前記ロックセットにロックセットペアを記憶させ、該ペアは、前記ロックセットの現在のメンバーと、前記ロックセットのメンバーでない前記新しいロックとの各独自の組み合わせであり、各ペアに対して、前記ロックセットの現在のメンバーが第1の位置に記録され、前記新しいロックが第2の位置に記録されるか、または、前記ロックセットの現在のメンバーが第2の位置に記録され、前記新しいロックが第1の位置に記憶されるか、するように決定されたオーダーにて前記ペアを記憶させることを含む請求項1記載の方法。

【請求項4】 前記ロックログのペアは、順序付けされたロックグラフとして表されており、さらに、各ロックのためのノードと、第1のロックを第2のロックに接続する方向付けエッジとを備えており、該エッジの方向は、前記第1および第2のロックが獲得されたオーダーを示しており、第1のノードおよび第2のノードが反対方向に向けられた第1のエッジおよび第2のエッジによって接続されるようなトポロジカルソートを前記ロックグラフに対して行い、前記第1および第2のノードに関連したロックをコンカレントロックエラーとして報告することを含む請求項3記載の方法。

【請求項5】 ヒープの各アドレスに対して、メモリアロケーションおよびメモリアロケーション命令を捜し出し置き換えることによってシャドーヒープにおけるシャドーアドレスを維持し、前記ヒープのアドレスのデータにアクセスする命令を捜し出し、該アクセス命令をモ

ニタリングプロシージャに対する命令で置き換え、前記モニタリングプロシージャによって対応するシャドーアドレスにロックセット情報を記録し、前記アクセス命令によるアクセスが前記記録されたロックセット情報と一致しない場合に、一致しないコンカレント状態をデータ競合状態として報告することを含む請求項1記載の方法。

【請求項6】 前記ヒープのアロケートされたアドレスへのアクセスを検出することを含む請求項5記載の方法。

【請求項7】 現在アクセスしているスレッドの現在のロックセット情報を前記ヒープのアドレスにアクセスしている前のスレッドの記憶されたロックセット情報でインターセクトし、前記セットのインターセクションが空である場合に、データ競合状態を報告し、現在のロックセットと前のロックセットとが異なる場合に、現在のロックセット情報と記憶されたロックセット情報とのインターセクションを記憶し、前記アクセスが前記アドレスへの書き込みである場合に、前記現在のロックセットと記憶されたロックセットとのインターセクションを書き込むことを含む請求項6記載の方法。

【請求項8】 前記スレッドアクセスが共用されないときに、アクセスするスレッドのアイデンティティを記憶することを含む請求項5記載の方法。

【請求項9】 読み取りアクセスに対して読み取りロックセットを維持し、書き込みアクセスに対して書き込みロックセットを維持し、読み取りアクセスに対して前記読み取りロックセットに読み取りロックを加え、書き込みアクセスに対して前記書き込みロックセットに書き込みロックを加えることを含む請求項1記載の方法。

【請求項10】 前記ロックセットをアレーのエントリに記録し、特定のエントリに記憶されたロックセットに関連した各シャドーアドレスでのアレーのその特定のエントリに対するポインタを記憶させ、特定のロックセットからハッシュ値を決定し、該ハッシュ値によって決定されるようなハッシュテーブルに前記ポインタを記憶させることを含む請求項5記載の方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、一般的には、コンピュータシステムにおいてプログラムスレッドをコンカレントに実行することに関し、より詳細に述べるならば、コンカレントエラーを検出することに関するものである。

【0002】

【従来の技術】コンプレックスコンピュータシステムにおいては、処理をスピードアップするために、多重実行パスまたは“スレッド”が使用されうる。例えば、ユーザとの対話は、フォアグラウンドスレッドによって取り扱われうるが、一方、データベース更新の如き計算集中

オペレーションは、バックグラウンドスレッドによって取り扱われる。また、性能を改善するために、パラレルスレッドが多重プロセッサにて実行されうる。

【0003】コンカレントエラーを検出することが問題である。一つの型のコンカレントエラーは、“デッドロック”、すなわち、デッドロック状態である。例えば、スレッドAが共用リソースXを所有し、また、リソースYを獲得することを必要とするような場合に、デッドロック状態が存在する。リソースYは、リソースXを獲得することを必要としている第2のスレッドBによって所有されている。この結果、リソースのコンフリクトが生ずる。典型的には、デッドロック状態により、システムの意図された機能が停止させられてしまう。

【0004】もう一つ別の型のコンカレンシー問題は、競合状態である。データ状態がスレッド実行のスピードおよびオーダーに依存する場合に、競合状態が存在しうる。例えば、あるスレッドは、他のスレッドによって書き込まれるアドレスのデータを読み出す必要があるかもしれない。読み出される実際のデータは、個々のスレッドにより読み取りおよび書き込みのオーダーに依存する。このような競合状態から生ずる非決定論により、そのプログラムはエラーを含む結果を生じてしまう。

【0005】知られている従来の技術は、コンカレントエラーを静的にチェックするものである。典型的には、従来の技術は、プログラムが正しい、すなわち、“真”であるという定理を立証している。

【0006】厳密なロックオーダーまたはデータアクセスシーケンスは、スレッドが実行されている間に、動的に変化することがあるということが問題である。このような変化は、静的チェッカーでは検出し得ない。したがって、コンカレントエラーを動的にチェックすることが望まれている。

【0007】

【発明の概要】本発明によれば、プログラムにおけるコンカレントエラーを検出するためのコンピュータによって実施される方法が提供される。多重プログラムスレッドの機械語は、スレッドのコンカレント状態をもたらす命令を捜し出して置き換えるようにインストールされる。置換命令は、モニターのプロシージャをコールする。

【0008】本発明は、上位概念において、本特許請求の範囲の請求項1に限定されたような、プログラムにおけるコンカレントエラーを検出するためのコンピュータによって実施される方法にある。

【0009】後述するように、モニタープロシージャは、多重スレッドが実行されている間、メモリにコンカレント状態情報を記録する。その記録されたコンカレント状態情報は、分析され得て、一致しないコンカレント状態遷移は、コンカレントエラーとして報告され得る。

【0010】置換される命令は、ロックを獲得し解放す

る命令であるのが好ましい。この場合において、記録された状態情報は、ロックのロックオーダーである。ロックオーダーにおけるサイクルは、デッドロック状態を示している。

【0011】ヒープのアドレス毎に、シャドーアドレスが、シャドーヒープに維持される。これは、メモリアロケーションおよびデアロケーション命令を捜し出し置き換えることによって行われうる。この場合において、記録された状態情報は、データがアクセスされるオーダーに関連付けられている。一致しないアクセスオーダーは、データ競合状態として報告される。

【0012】

【発明の実施の形態】次に、添付図面に基づいて、本発明の実施の形態について、本発明をより詳細に説明する。

【0013】図1は、多重プログラムスレッドのコンカレントエラーを動的に検出するのに使用されうるプロセス100を示す。プログラマーは、通常のプログラミング方法を使用してソースプログラム110を作成する。ソースプログラム110は、処理されるとき、コンピュータシステム(CPU)190において同時に実行されるようになっている。コンピュータシステム190は、1つのメモリおよび1つのプロセッサを含むか、関連して使用される複数のメモリおよびプロセッサを含む。プログラム110は、コンパイラ111によってオブジェクトモジュール(thread obj)120へとコンパイルされる。

【0014】ソースプログラム110が実行されている間そのソースプログラムの挙動に関して報告を行なうためのモニタープログラム130も作成される。ソースプログラム110の実行に関してモニタープログラム130が如何にして報告を行なうことができるかについての詳細については、後述する。モニタープログラム130は、コンパイラ131によってオブジェクトモジュール(mon obj)121へとコンパイルされる。リンカー140は、オブジェクトモジュール120-121を機械イメージ(.exe)150へとリンクする。機械イメージ150は、“バイナリー”プログラムまたは“テキスト”として知られている。スレッドの実行に関する報告をするためのイメージ150の統合部分は、“モニター”155と称される。

【0015】ここで注意すべきことは、リンカー140は、コンパイラ111および131によって作成されたシンボルテーブル141にアクセスするというのである。シンボルテーブル141を使用して、リンカー140は、例えば、オブジェクトモジュールの間で共通なプロシージャコールおよびデータレファレンスを分析しうる。また、ここで注意すべきことは、プログラムのコンパILINGおよびリンキングにより、プログラムのテキスト部分150と共に通される定数値を記憶するデ

ータセクション (data) 151が発生されうるということである。一つの実施例として、プログラムテキストまたは機械イメージ150は、機械命令、例えば、オペランドおよびオペレータのみを含む。すべてのデータは、データセクション151に別々に記憶される。

【0016】機械イメージ150は、インストルメンター160によって変更され、すなわち、“パッチ”される。インストルメンテーションは、イメージ150の選択された命令を捜し出し置き換えるプロセスである。使用されうるインストルメンターの例は、1996年7月3日にShrivastava氏等に与えられた米国特許第5539907号「System for Monitoring Computer System Performance」に開示されている。

【0017】各選択された命令は、1つまたはそれ以上の置換命令でオーバーライトされる。例えば、置換命令は、サブルーチンへの飛び越し命令 (jsr) 171でありうる。jsr命令は、モニター150のプロシージャ173のうちの一つを“コール” (172) することができる。プロシージャ173は、選択された命令がまさに実行されようとしている時の、システムのコンカレント状態を調べ、記録し、分析するように構成されている。

【0018】別の実施例において処理をスピードアップするために、コール172は、インライン命令として実施されうる。すなわち、インライン命令は、プロシージャ173を実施する。したがって、以下の説明においては、コール172は、呼び出されるルーチンのサブルーチンコールまたは等価なインライン命令を意味している。

【0019】機械語150がインストルメントされた後、この機械語は、システムのメモリに記憶される。このために、インストルメンテーションは、“バイナリリライト”として知られている。

【0020】変形機械語150は、ローダー180によってプロセッサ190へロードされうる。プログラムを実行している間に、挿入された“コール”172は、スレッド実行をインタセプトする。プロシージャ173は、システムについての選択されたコンカレント状態情報195を記録する。記録された状態情報は、それら状態に関してプログラムの挙動を報告196するために分析されうる。

【0021】例えば、一致しないオーダーにおけるロッキング、または、一致しないオーダーにおけるデータの記憶は、コンカレントエラーを反映していることがありうる。モニタープロシージャ173の完了後、置換された命令が実行でき、プログラムスレッドの通常の実行は、サブルーチンからの復帰命令174を介して再開されうる。コンカレントエラーは、プログラムが実行されている間、または、モニタリングの完了後に報告されうる。

【0022】インストルメンテーション中に置き換えら

れる特定の命令171は、どのスレッドコンカレント状態を分析し報告すべきかに依存している。デッドロック状態を検出するために、ロックを獲得し解放させるのに使用されるプロシージャをコールするすべての機械命令がインストルメントされる。競合状態を検出するために、ロックを獲得し解放させるのに使用されるプロシージャをコールするすべての機械命令がインストルメントされ、それに加えて、データアクセス命令、例えば、ロードおよびストア、並びに、メモリアロケーションおよびデアロケーションコールもインストルメントされる。

【0023】図2を参照して、デッドロックコンカレントエラーの検出について説明する。システム190において、複数のスレッド (A、B、C) 211-213が命令をコンカレントに実行している。スレッド211-213は、それらのアクティビティ、例えば、共用リソースへのアクセスを同期するためロック (a、b、...、e、f) へアクセスする。

【0024】所定のアクティビティを同期するために、スレッドは、関連するロックを獲得するためにコール221をなす。そのロックをホールドすることにより、そのスレッドは、他のスレッドが同じ共用リソースを使用しないようにすることができる。そのアクティビティが完了したときに、コール222がそのロックを解放する。単一のコールが複数のロックを獲得し、または解放することもありうる。マニピュレートされる特定のロックは、コール221-222のアーギュメントとして通されうる。もし、所望のロックが使用できない場合、例えば、そのロックが他のスレッドによってホールドされている場合には、その獲得しようとしているスレッドは、そのロックが使用できるようになるまで、待ち状態にホールドされうる。

【0025】デッドロック状態を検出するための本発明の方法によれば、ロックが正しく獲得されるべきオーダーが予め規定される。これは、マルチプロセスおよびマルチスレッドプログラムにおけるロックの獲得をマネージするための普通のディシプリンである。反例として、スレッドAは、ロックaをホールドし、スレッドBは、ロックbをホールドする。前に進むためには、スレッドAもロックbを必要とし、スレッドBは、ロックaを必要とする。獲得オーダーが競合するこのような場合には、どちらのスレッドも前に進めない。これは、デッドロックのクラシカルなケースである。

【0026】コンプレックス実行環境においては、処理アクティビティを同期するために数千のロックが使用されうる。一致したロッキング挙動は、ある程度までは、静的にモニターされうる。もし、ロックが獲得されるオーダーが特定の実行に依存している場合には、デッドロック状態を検出することは非常に難しいものとなる。何故ならば、ことなるロッキングオーダーにおける置換の数は、非常に大きくなるからである。

【0027】したがって、図1のモニター155は、スレッドが実行されている間におけるロックが獲得され解放されるオーダーに関連したコンカレント状態情報を記憶するデータ構造を維持する。この状態情報は、プログラムが一般的な使用のために利用できるようにされる前に、ロッキングオーダーにおけるエラーを検出するために分析されうる。

【0028】もし、インストルメンター160がマネージメントプロシージャをロックするすべてのコール171を捜し出しパッチする場合には、1つのロックが獲得され、または、解放される毎に、それらスレッドの実行フローは、図1のプロシージャ173の1つによってインタセプトされうる。

【0029】より詳細に述べるならば、モニターは、各スレッドに対して1つのロックセット230を維持する。1つのロックが獲得されるときはいつでも、そのロックは、そのスレッドに関連したロックセット230のメンバーとなる。そのロックが解放されるとき、そのロックは、そのセットから削除される。例えば、もし、スレッドAがロックa、b、cを獲得し、後でロックbを解放する場合には、そのロックセットは、メンバー{a、c}を有する。

【0030】さらにまた、モニターは、ロックログ300にすべてのロック獲得を記録する。図3に示されるように、獲得されたロックは、順序付けされたペア301-312として次のように記録される。例えば、スレッドAは、順に、ロックa、b、cを獲得する。ロックbが獲得されるとき、そのペア(a、b)301は、そのログに記録される。これは、ロックbがロックaの後で獲得されたことを示している。その後、スレッドがロックcを獲得するとき、それらペア(a、c)および(b、c)もまた記録される。何故ならば、ロックaおよびbは、共に、ロックcが獲得されたときでも、スレッドAのロックセットのメンバーであるからである。

【0031】そのプログラムの実行のモニタリング中にロッキングオーダーが変更されない場合には、そのオーダーリング関係を利用する最適化が可能である。このような最適化において、1つのスレッドによってホールドされるロックのセットは、最も前に獲得されたものから、最も最近に獲得されたものまで、時間的順序にて維持される。このような最適化でもって、新しいロックがあるスレッドによって獲得されたときには、最も最近に獲得されたロックおよび今獲得されつつあるロックからなるペアがログに記憶される。

【0032】したがって、一般的な場合において、あるスレッドがあるロックを獲得するときにはいつでも、あるペアが、そのスレッドのロックセットの現在のメンバーとその新しく獲得されるロックとの組合せ毎に、ロックログ300に記憶される。換言するならば、それらペアは、ロッキングのオーダーを表している。もし、ロック

ログ300が既に同一ペアを記録している場合には、その新しいペアは、記録されない。すなわち、そのログは、独自のペアのみを記憶する。ことなるオーダーにて獲得されたロックを表す2つのペア、例えば、(a、b)と(b、a)とは同一ではない。

【0033】図3の例であるログ300は、ロックa、cおよびdを獲得しているスレッドBを示している。その後で、スレッドAは、ロックaおよびcを解放したが、ロックd、eおよびfを獲得した。それから、スレッドBは、ロックa、cおよびdを解放し、ロックfおよびbを獲得する。最後のロック獲得は、ログ300においてペア(f、b)312として反映されている。

【0034】図4は、図3のロックログ300の図式的表示400を示している。ここで、各ロックは、ロックオーダーリンググラフ400におけるノード401-403であり、ノードのペア間の方向付けエッジ404-405は、それらロックが獲得された相対的オーダーを示している。

【0035】ここで、この例のロッキングオーダーによれば、反対方向における2つの相互接続エッジ404および405を有するノード402および403が生ずることは明らかである。これは、サイクルと称される。サイクルは、スレッドが競合オーダーにてロックを獲得していることを示している。方向付けグラフにおけるサイクルは、ロックログ300のペアに関してトポロジカルソートを行うことによって検出されうる。

【0036】他のアプリケーションのためのトポロジカルソートについては、D.Knuth氏による「The of Programming, Volume I, Fundamental Algorithms, Addison-Wesley, 1968, pp.258-268」に一般的に開示されている。サイクルは、潜在的なデッドロック状態の証拠であり、したがって、報告される。

【0037】ここで注意すべきことは、ロックオーダーリンググラフ400にサイクルがないことは、そのプログラムが絶対的にデッドロックのないことを意味するものでないということである。何故ならば、他のリソース同期化機構がデッドロック状態に巻き込まれうるからである。さらに、スレッドが異なるオーダーにおいてロックを獲得するようにさせる入力データの別のセットもありうる。一般的に、コンカレントエラーを動的にチェックしても、そのプログラムの特定の実行についてスレッドが適切にロックすることを立証することができるだけである。

【0038】別の型のコンカレントエラーは、データ競合状態である。競合状態は、複数のスレッドが同一アドレスのデータにアクセスするときに起こる。例えば、2つのスレッドの両者が変更する意図をもってデータを読み取る。この場合において、最終データ状態は、どちらのスレッドが最初に完了するかに依存している。

【0039】大抵のオペレーティングシステムにおいて

は、スケジューラは、任意のオーダーにおいてスレッドを実行する自由を有しており、同期化の制約のみを受ける。マルチプロセッサシステムにおいては、実行は、例えば、異なるプロセッサのキャッシュミスおよびその他のロード条件に依存した異なるスピードで行われうる。したがって、もし、完了時間が動的に変化する場合に、最終データ状態も変化する。

【0040】もし、各共用データがロックで保護される場合には、データ競合は避けられうる。大きなプログラムの複数のスレッドの間では、どのロックがどのデータに関連しているのかを静的に確認することは難しい。

【0041】図5は、アドレスレベルでデータ競合状態を検出するのに使用されうるデータ構造を示している。ランダムアクセスメモリ(RAM)500は、“ヒープ”510のアドレスにデータを記憶する。ヒープ510は、データを共用するために複数のプログラムスレッドによってコンカレントにアクセスされうる。

【0042】データは、アドレス511の如き独自のアドレスに記憶される。これらアドレスは、仮想メモリアドレスでありうる。これは、ダイナミック物理メモリ

(DRAM)の総量がアドレスされうるものよりも少ないことを意味している。1つのありうる仮想メモリシステムにおいては、アドレス511は、32ビット値として表され得て、メモリの4ギガバイトのアドレッシングを可能とする。物理メモリとしては、そのうちの少量部分のみしか実施され得ない。ここで理解すべきことは、これは、多くの変化のうちの1つのみであり、例えば、64ビットワードしか使用され得ないということである。

【0043】ヒープ510のアドレス毎に、シャドーヒープ520もまた維持される。これは、アドレス511とアドレス521との間の1対1の対応501があることを意味している。シャドーヒープ520は、アクセスコンカレント状態情報を記憶するのに使用される。これは、ヒープのあるアドレスのデータがアクセスされるときはいつでも、アルゴリズムがそのように要求する場合には、現在のデータアクセスコンカレント情報は、シャドーヒープ520の対応するアドレスにて更新される。

【0044】図1の機械語150は、次のように変更される。ヒープ(ロードおよびストア)のアドレスにアクセスするすべての命令171は、モニター155のプロシージャ173へのコール172によって置き換えられる。さらに、ヒープ510のアドレスをアロケートおよびデアロケートするプロシージャへの機械語150によるコールは、手順をモニタリングするコールで置き換えられる。この後のアクティビティは、シャドーヒープ520がそのヒープ510のアドレスを正確に反映するようにするために行われる。

【0045】図6は、シャドーヒープ520のアドレスの1つにおけるワード600を示している。このワード

600は、2つのフィールド、すなわち、アクセス状態フィールド610と、ロックセット状態フィールド620とを有している。後述するように、ある特定の実施においては、状態フィールド620は、2つのサブフィールド621-622を含み得る。アクセス状態フィールド610は、1から3ビット、例えば、アクセス

(A)、共用(S)および読取り/書き込み(R/W)611、612および613を有しうる。

【0046】ワード600のアドレスは、ヒープ510の対応するアドレスがアロケートされるとき、シャドーヒープ520にアロケートされうる。ワード600は、すべてのビットを零へセットするように初期化される。これは、ヒープアドレスの対応するワードが決してアドレスされていないことを示している。ヒープにおけるワードのデアロケーション時には、そのシャドーヒープの対応するワードがデアロケートされ、それが以前に記憶していたどの情報も失われる。このデアロケーションは、ライブラリデアロケーションルーチンに対するプログラムにおける明示コールを通して起こりうるか、または、後述するように注釈されたガーベジコレクターの如き自動記憶リクレーションルーチンを通して起こりうる。

【0047】図7は、シャドーヒープ520を維持するのに使用されうるプロセス700を示している。図2の任意のスレッド211がステップ710でまさにヒープ510のアドレスのデータにアクセスしようとするとき、ステップ720において、シャドーヒープ520の対応するアドレスがアロケートされていたかを決定する。シャドーヒープアドレスがないことにより、ステップ725においてヒープ510のアロケートされていない部分にアクセスする誤った試みが検出される。

【0048】もし、妥当なシャドーヒープアドレスに記憶された値が零(ステップ730における以前のアクセスのない)である場合には、ステップ732において、状態フィールド610のアクセスビット711を論理“1”へセットし、ステップ735において、アクセスしているスレッドに関連した現在のロックセットをロックセットフィールド620へ書き込む。置換された命令を実行した後、そのスレッドの実行を再開する。ロックセットがどのようにしてシャドーヒープ520のアドレスに記憶されかについては、後で詳述する。後述するように実際には、ロックセット状態フィールド620は、そのロックセットが記憶されるアレー素子に対するポインタを記憶する。したがって、ロックセットの“書き込み”は、ワード600のポインタの記憶を意味している。

【0049】その後、同じアドレスに対する別のアクセスにより、アクセスビット711が零でないことが検出される。この場合において、アドレスにアクセスしているスレッドのロックセットとシャドーアドレスに記憶さ

10

20

30

40

50

れたロックセットとのインターセクションをとり、ステップ740において残りのロックセットをロックセットフィールド620に記憶する。もし、インターセクションをとった後ロックセットフィールド620がステップ750において空である場合、例えば、共通の未決定のロックがない場合には、ステップ755においてそのアドレスに対するデータ競合状態がありうるか、または、さもなければ、戻る。アドレスをデアロケートすることにより、ワード600は、その初期のすべてゼロ値へとリセットさせられる。

【0050】この方法は、データ競合状態を誤って合図する可能性がある。これらは、誤りアラームとして知られている。誤りアラームは、単一のスレッドのみがデータをアクセスしている場合に起こりうる。

【0051】図8は、プロセス700がどのようにしてより弁別しうるコンカレント検出を行うように修正されるかを示している。この変形においては、もし、アクセスビット711が非零である場合に、付加的なチェックが共用ビット712に関してなされる（ステップ760）。共用ビット712が零である間、スレッド識別子が、シャドーアドレスのフィールドに記憶される。

【0052】その後のアクセスにおいて、もし、共用ビット712がなおも零である場合には、アクセスしているスレッドの識別子が、シャドーアドレスに記憶された識別子と比較される。もし、それら識別子が異なる場合には、共用ビット712は、少なくとも2つのスレッドがそのアドレスにアクセスしたことを示すために、論理“1”にセットされる。

【0053】ロックセットは、共用ビット712が“1”へセットされるまでは、シャドーアドレスに記憶されない。何故ならば、そのアドレスが1つより多いスレッドによって共用されるまでは、エラーチェックを行う必要がないからである。スレッド識別子およびロックセットは同時にシャドーアドレスに記憶される必要はないので、シャドーアドレスの同じフィールド620が、異なる時間にてこれらの値を記憶するのに使用される。

【0054】共用ビット712がゼロのままである限り、そのプロセスは、ステップ735においてシャドーアドレスのロックセットフィールド620にそのスレッドのアイデンティティを単に書き込む。この場合において、ロックセットは、同じスレッドがそのアドレスのデータにアクセスし続ける限り、もはやインターセクトされない。もし、共用ビット711が“1”にセットされる場合には、空のセット740および750についてインターセクトし、チェックする。

【0055】しかしながら、この方法は、あるアプリケーションに対しては粗過ぎて、なおも誤りアラームを生ずる可能性がある。例えば、多くのデータ処理システムにおいては、多くのスレッドがその後にこれらのアドレ

スを読み取ることがあるとしても、多くのアドレスが1つのスレッドによって一度初期化されるが、決して再び書き込まれない。この場合において、ロックは、それが最後に書き込まれたときに、そのアドレスを保護するために必要とされない。したがって、前述したコンカレントチェック方法は、多くの誤りアラームを報告する可能性がある。

【0056】シャドーアドレスに記憶されたR/Wビット613は、この問題を克服するのに使用される。このビットは、共用ビット612がセットされるまでは、無視される。共用ビット612がセットされるとき、R/Wビット613は、次の書き込みアクセス時に“1”にセットされる。もし、共用ビット612が書き込みアクセスによりセットされる場合には、R/Wビット613もまたすぐにセットされる。

【0057】シャドーアドレス600に記憶されたロックセット620に対する更新は、R/Wビット613の状態とは関係なく、通常のように進められる。R/Wビット613の作用は、ビットが零であるときに、エラーメッセージを抑圧することである。何故ならば、少なくとも2つのスレッドがあるアドレスのデータにアクセスしておらず（共用ビット612は“1”にセットされる）且つそのアドレスの共用後にそれらスレッドの少なくとも一方によってそのアドレスが書き込まれていない（R/Wビットが“1”にセットされる）限り、アクセスエラーは起こり得ないからである。

【0058】ある場合に使用されうる1つのプログラミング取決めは、1つより多いロックで単一のアドレスに記憶されるデータを保護する。この取決めにおいては、スレッドは、データが変更される前に、すべてのロックを獲得しなければならないが、1つのスレッドは、そのデータを読み取るために1つのロックを必要とするだけである。本発明の方法は、このようなプログラミング取決めを受け入れるように変更されうる。ある書き込みアクセス時に、そのプロセスは、前述したように、シャドーアドレスを更新しうる。ある読み取りアクセス時に、空セットに対するロックセットインターセクションおよびテストが前述したように行われる。しかしながら、この場合において、シャドーアドレスに記憶されたロックセットは、インターセクトされたセットで更新されない。

【0059】このコンカレントエラー検出法は、ロックが2つのモード、すなわち、読み取りモードおよび書き込みモードのうちの一方において獲得されうるような単一ライター/多重リーダーロックを取り扱うように更に改良されうる。もし、あるスレッドが書き込みモードにおいてあるロックを獲得する場合には、そのスレッドは、他のスレッドがどのような場合でもロックをホールドしないように保証される。もし、あるスレッドが読み取りモードにおいてそのロックを獲得する場合には、他

のスレッドが書き込みモードにおいてロックをホールドしないことが保証されるだけである。

【0060】単一ライター／多重リーダーロックにより、多重スレッドがあるアドレスに記憶されたデータをコンカレントに読み取れるようにされる。これは、多重ロックでアドレスを保護する概念に類似している。しかしながら、実施のための考え方は、本発明のコンカレントエラー検出法においてロックを受け入れる仕方のように、異なる。

【0061】このような説明においては、単一のロックセットのみが1つのスレッドに関連付けられている。ロックがそのスレッドによって獲得され解放されるときに、ロックはそのセットに加えられ、そのセットから除かれる。単一ライター／多重リーダーロックを受け入れるために、2つのロックセット、すなわち、読み取りセットおよび書き込みセットが各スレッドに関連付けられる。

【0062】ある書き込みロックが獲得されるとき、それは、両方のセットに加えられ、そのロックは、そのロックが解放されるときに、それら2つのセットから除かれる。読み取りロックの場合には、そのロックが読み取りセットに加えられ、読み取りセットから解放されるだけである。シャドーアドレスでの情報の処理は、読み取りアクセス中に読み取りセットのみが使用され、書き込みアクセス中に書き込みセットが使用されること以外は、前述したものと同様である。

【0063】次に、ロックセットおよびロックセットに対するポインターをオルガナイズするための好ましい仕方についてより詳細に説明する。図9は、複数のロックセット901-907を含むアレー900を示している。このアレー900における各エントリは、ロックの独自のセットである。実際には、異なるロックセットの数は、小さく、したがって、テーブル900は、大抵の場合において、メインメモリに記憶されうる。テーブル900のエントリは、特定のモニタリングセッション中において恒久的なものである。これは、それらエントリが変更されず、テーブル900から削除されないことを意味している。

【0064】識別子として使用される小さな整数値が各ロックセットに関連付けられる。この識別子は、アレー900へのインデックスである。テーブル900におけるエントリは、エントリ“0”から始まって順次、エントリ“1”、・・・等々の如く発生されていく。アレーへのインデックスは、そのエントリの識別子である。図6のロックセット状態フィールド620に実際に記憶されるのは、これらの識別子である。

【0065】テーブルを小さく保つために、一つのエントリは、1回より多くはアレー900に加えられない。したがって、補助ハッシュテーブル1100が維持される。一つのロックセットエントリがアレー900

に発生される毎に、ハッシュテーブルエントリ1110も発生される。ハッシュキー値は、ハッシングファンクションをロックセットに直接に適用することによって導出される。ロックセット識別子（アレーインデックス）は、対応するハッシュテーブル値に記憶される。したがって、オペレーション中、ロックセットは、独自に識別されうる。

【0066】デッドロックを検出する方法の場合のように、競合状態を検出するためのこれらの方法の使用により発生される結果は、モニターされる特定のプログラム実行に依存している。異なる入力データまたは事象のわずかに異なるシーケンスにより、プログラムを通しての実行パスが異なってくることがある。このために、この方法を使用するときには、起こりうる競合状態のすべてが検出されるというわけにはいかないことがありうる。実際に同期化がロッキング以外のあるメカニズムにより起きたときや、競合状態が実際に起きえないときに、潜在的競合状態が報告されることがありうる。

【0067】さらに改善したものとして、チェックされるべきプログラムに注釈を付けることができる。プログラム注釈は、プログラマーの意図を示すためにそのプログラムに付される“シグナル”である。注釈は、プログラムの実行に影響を与えないものである。一つの効果として、注釈は、コンカレントエラー検出プロセスを改良するために実行中にチェックされうる。

【0068】ソース注釈の例としては、所定のアーギュメントでの“ナル”プロシージャへのコールの形でありうる。このナルプロシージャは、その名が意味するように、正規の実行フローへの戻りにほかならない。ナルプロシージャへのコールは、インストルメンター160によって識別されうる。これらのコールは、アーギュメントを解釈するモニター155のプロシージャ173へのコールによって置き換えられうる。別の仕方として、注釈は、モニターのプロシージャへのコールの形でもありうる。これらのプロシージャは、シンボルテーブル141にて識別されうる。

【0069】例えば、ある注釈は、次のアロケーションが読み取り命令によって共用モードにてアクセスされるだけのものであるメモリアドレスに対するものであることを示すことがありうる。別の注釈は、アロケートされるメモリがそのスレッドに対して専用であることを示すことができる。明らかなように、このような注釈は、誤りアラームの数を減少させるものである。

【0070】デッドロック検出の場合に加えられうる注釈の別の例として、プログラムに使用されるロッキングオーダーが変更されていることを示す注釈がある。例えば、このような注釈は、ロッキングオーダーにおけるロックの位置が変更され且つそれを含むすべてのペアがログから除去されるべきであったことを示すことがありうる。

【0071】動的競合検出の場合において、もし、プログラムが自動記憶リクラメーション（ガーベジコレクション）を与える場合には、モニターに対するコールの形の注釈は、記憶のあるレンジがリクラメートされており、シャドーヒープからデアロケートされるべきであることを示すために、プログラムのソースに含まれる。

【0072】本発明の特定の実施例について説明してきたのであるが、これらの実施例の種々な変形態様が考えられ、これら変形態様においても、前述したような効果のすべて、またはそれら効果のうちのいくつかを達成しうことは明らかであろう。したがって、本請求の範囲の記載は、本発明の範囲内に入るような変形態様のすべてを包含しようとするものである。

【図面の簡単な説明】

【図1】プログラムにおける同期エラーを動的に検出するためのプロセスを示すブロック図である。

【図2】多重スレッドおよびロックセットを示すブロック図である。

【図3】ロックログを示すブロック図である。

【図4】ロックオーダーリンググラフを示すブロック図である。

【図5】ヒープおよびシャドーヒープを示すブロック図である。

【図6】ロックセットを示すブロック図である。

【図7】コンカレントアクセスエラーをチェックするためのプロセスを示すブロック図である。

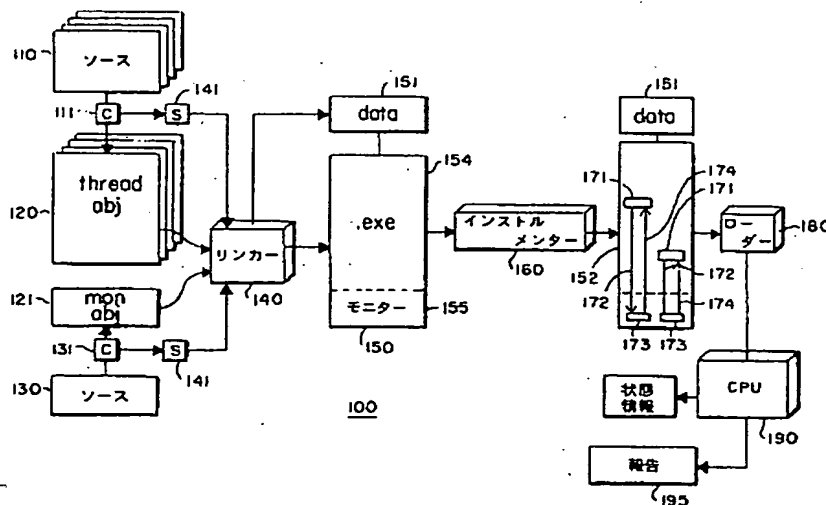
【図8】コンカレント共用アクセスエラーをチェックするための変形フローを示す図である。

【図9】ロックセットアレーおよびハッシュテーブルを示すブロック図である。

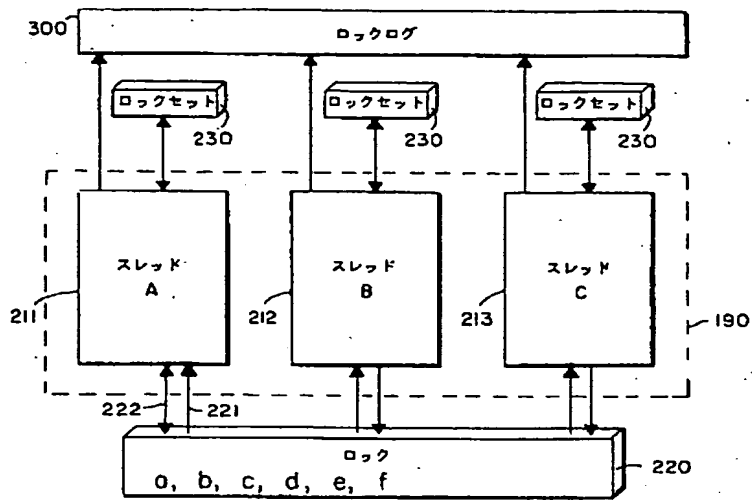
【符号の説明】

- 100 プロセス
- 110 ソースプログラム
- 111 コンパイラ
- 120 オブジェクトモジュール
- 121 オブジェクトモジュール
- 130 モニタープログラム
- 131 コンパイラ
- 140 リンカー
- 141 シンボルテーブル
- 150 機械イメージ
- 151 データセクション
- 155 モニター
- 160 インストルメンター
- 171 サブルーチンへの飛び越し命令
- 172 コール
- 173 プロシージャ
- 174 サブルーチンからの復帰命令
- 180 ローダー
- 190 プロセッサ
- 195 コンカレント状態情報

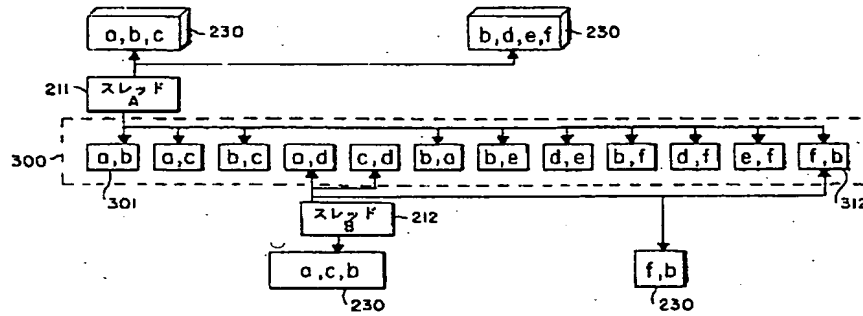
【図1】



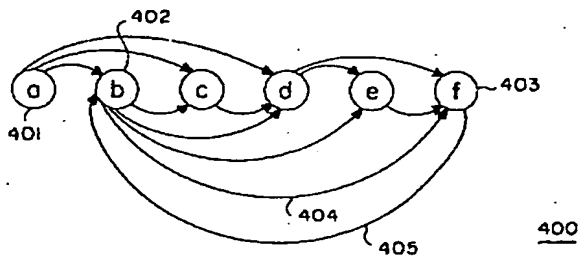
【図 2】



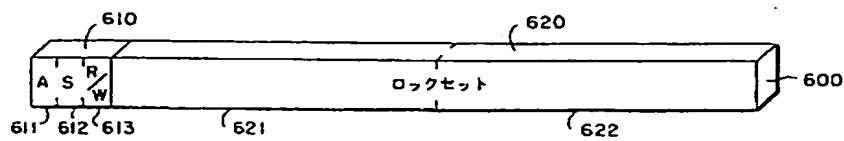
【図 3】



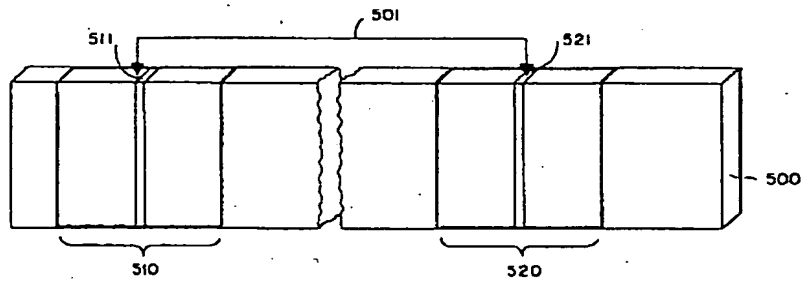
【図 4】



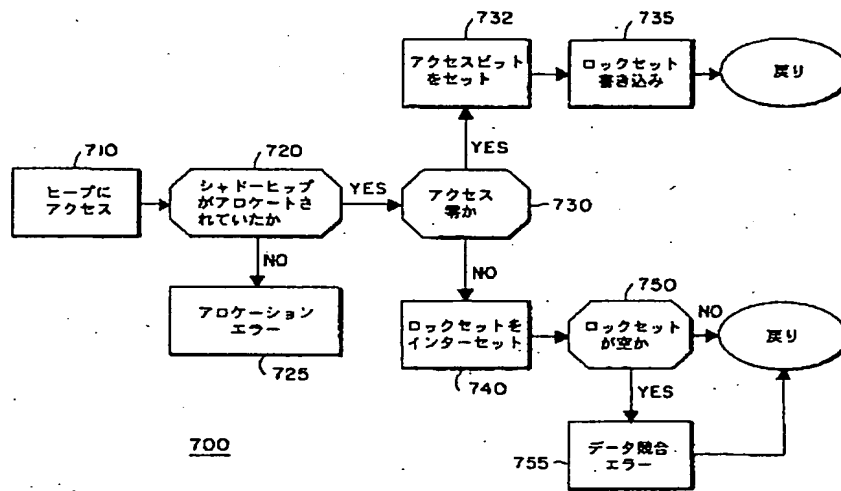
【図 6】



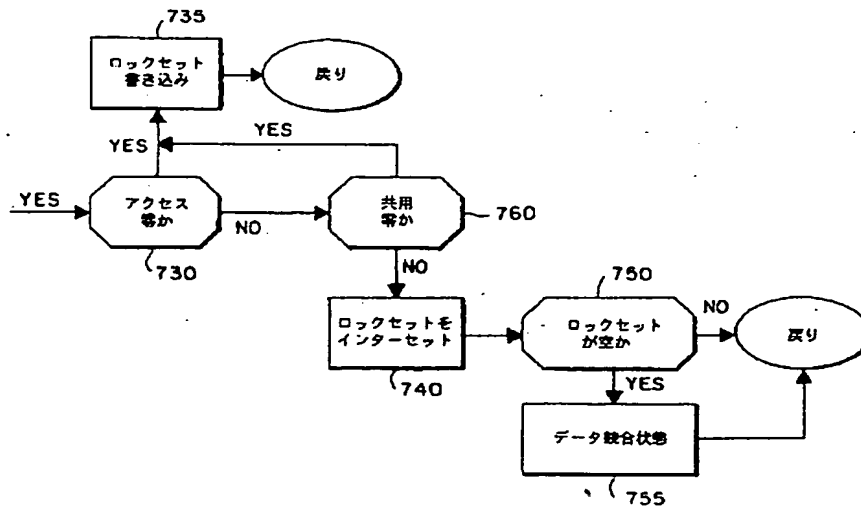
【図5】



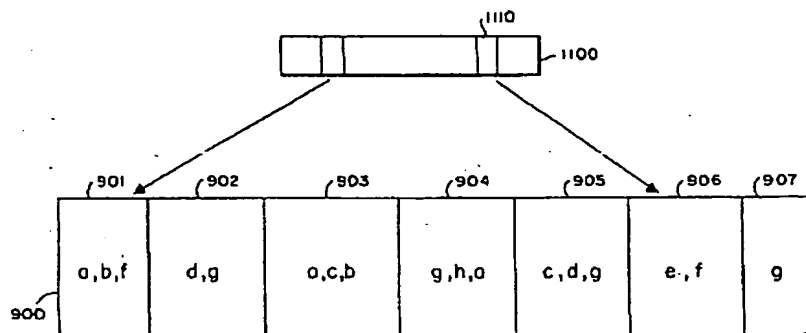
【図7】



【図8】



【図9】



フロントページの続き

(72)発明者 チャールズ ジー ネルソン
 アメリカ合衆国 カリフォルニア州
 94306 パロ アルト カレッジ アベニ
 ュー 770

(72)発明者 ステファン サヴェージ
 アメリカ合衆国 ワシントン州 98052
 レッドモンド ノースイースト ワンハン
 ドレッズ コート 17831
 (72)発明者 パトリック ジー ソバルヴァーロ
 アメリカ合衆国 カリフォルニア州
 94301 パロ アルト エマーソン スト
 リート 312